

N89-16304

225-61
167049
8P

A PROPOSED CLASSIFICATION SCHEME FOR ADA-BASED SOFTWARE PRODUCTS

Gary J. Cernosek
McDonnell Douglas Astronautics Co. - Houston
16055 Space Center Blvd.
Houston, Texas 77062
(713) 280-1500

1.0 INTRODUCTION

As the requirements for producing software in the Ada* language become a reality for projects such as the Space Station, a great amount of Ada-based program code will begin to emerge. Although this software will exist in Ada source code form, it will display varying degrees of quality based on the manner in which it was developed. In spite of the fact that Ada supports the most modern and effective concepts of programming available, poorly written programs can be created in Ada just as they have been in previous languages.

Consequently, the term "written in Ada" could have many connotations. The mere fact that a program exists in Ada source code form does not imply to any degree that there is any more quality in that product than would be if it were written in FORTRAN or C. If the modern features of the Ada language are not utilized to support the principles of software engineering, then the entire motivation and justification for moving to the Ada language will be defeated.

Recognizing this potential for varying levels of quality to result in Ada programs, what is needed is a classification scheme that describes the quality of a software product whose source code exists in Ada form. This classification assessment would be based on the overall process in which the software was developed, as well as the characteristics and attributes associated with the resulting source code produced. This provides an "after the fact" evaluation, and thus will not directly support proper development. However, the knowledge of the classification scheme may help in deterring bad development approaches and indirectly increase the overall quality consciousness of Ada-based software development.

This paper proposes a 5-level classification scheme that attempts to decompose this potentially broad spectrum of quality of which Ada programs may possess. The number of classes and their corresponding names are not as important as the mere fact that there needs to be some set of criteria from which to evaluate programs existing in Ada. An exact criteria for each class is not presented in the paper, nor are any detailed suggestions on how to effectively implement this quality assessment. The paper is merely intended to introduce the idea of Ada-based software classification and to suggest a set of requirements from which to base further research and development.

* Ada is a trademark of the U. S. Government (AJPO)

B.4.7.1.

ORIGINAL PAGE 18
OF POOR QUALITY

2.0 THE NEED FOR A CLASSIFICATION SCHEME

The purpose of the Ada language can be viewed from two perspectives. Technically, Ada was designed to strongly support the goals and principles of software engineering. However, the main influence driving the definition of Ada was economical. The "software crisis" was recognized in the early 1970's and the major cost factors were identified in software maintenance activities. Therefore, Ada was designed to give the potential for reducing software costs. Cost reductions start by providing a common language that consequently requires less compiler development and less programmer re-training. And as the amount of Ada code developed increases, the re-use of verified software components can further decrease development expenses.

Since the discipline of software engineering focuses on both technical and economic issues, the Ada language must be used as a software engineering tool and not merely as another programming language. Ada will not automatically meet its purpose and goals - it has to be used as it was designed to be used.

Therefore, it is unrealistic to expect that all software projects developed in Ada will realize the many benefits that the language has to offer. This is true not because the language is deficient, but rather because there are many different approaches to using any language. Several reasons why Ada may not be properly used on initial projects are outlined below:

Technical - The education and training required to learn how to effectively use Ada may be significant, especially for individuals without previous exposure to higher-level languages. Ada quality may suffer by having improperly trained personnel pre-maturely work on Ada development efforts.

Economical - The initial costs involved in moving to any new language are high. This characteristic may drive decision makers to short-term solutions, such as code translation approaches.

Political - Many organizations feel they are "locked" into a particular programming language, and often the machines that run their software. Even when Ada is shown to be technically superior and actually cost-effective, political influences can stifle attempts to upgrade an outdated software development environment.

Inertial - It is only natural for organizations to be reluctant to change. Ada, as well as other advances in computer engineering such as distributed processing, may intimidate people who feel more comfortable with their present environment. This natural state of inertia should be accepted and effectively dealt with rather than be a front line for personal battles.

(With these issues and many more to contend with, it is obvious that most organizations will have to transition into an Ada environment. As this transition is taking place (and possibly thereafter), a varying degree of quality must be expected to result among different development efforts. One way to measure the progress of transition is to classify the quality of the Ada software resulting from these efforts. The goal must be set to produce only the highest level of quality in Ada software. However, the reality must be recognized that it will be difficult to meet this goal in initial projects.

The suggested approach is to get started with Ada and do the best job possible under whatever circumstances may exist. The previously described road blocks should not prevent the exploration of Ada. However, the learning curve must be steep and be based on good sources of Ada training and education. Poor development habits must be broken and good ones must be created and enhanced. And most importantly, engineers and managers have to encourage the training and use of Ada. Without both peer-level and management support, effective transition to Ada will be difficult.

The most important theme to understand and constantly keep in mind is that the basis for "good" and "bad" rest in the goals and principles of software engineering. Software engineering represents the stable point of professional programming that can separate quality standards from personal style and allows concentration on issues above the language level.

Therefore, in order to measure the progress of transitioning to Ada, a software engineering-based classification method is needed. This is also in accordance with the DOD-STD-2167 Software Documentation Standard, which has changed the emphasis on Quality Assurance to Quality Evaluation. A proposed classification scheme for evaluating Ada software quality is presented in the next section.

ORIGINAL PAGE IS
OF POOR QUALITY

3.0 CLASSIFICATION METHOD AND CRITERIA

Each of the classifications below are described with the following format:

- O Classification level number: 5 (lowest) to 1 (highest)
- O Development Process Statement - phrase that references the approach taken in development:
 - oo Level 5 - "Translated To Ada"
 - oo Level 4 - "Coded In Ada"
 - oo Level 3 - "Programmed In Ada"
 - oo Level 2 - "Designed Into Ada"
 - oo Level 1 - "Engineered With Ada"
- O Description of the process in which the program source code was created
- O Characteristics and attributes indicative of the particular level of quality

Level 5 - "Translated To Ada"

This lowest class of Ada software implies nothing more than the fact that the program code exists in Ada form. The Ada code is created by some type of code translation, either through a manual and direct mapping performed by a human coding specialist, or by an automated code translator. Level 5 classification is intended for programs that have been previously developed in another language and have been converted to Ada merely to meet a requirement for the software to exist in Ada. However, programs that have been properly re-structured or re-designed into Ada have potential for a higher quality assessment.

The characteristics of Level 5 software include significant maintenance problems due to lack of readable and understandable code. None of the aesthetic qualities of the Ada language are evident due to the absence of human engineering. Additionally, the overall program structure is characteristic of the original language's form and represents the most inappropriate and ineffective use of the Ada language. A possible exception to this evaluation is when an organization wants to escape the previous language environment and allow 100% of its future development and maintenance in Ada.

Level 4 - "Coded In Ada"

Although Level 4 programs are humanly written in Ada, they lack the basic quality characteristics possible in good Ada programs. The development process is generally based on program development personnel that are not properly trained in utilizing the Ada language and its support environment properly and effectively.

The approach to development is ad hoc with no basis on formal software requirements definition and no documented design process. Level 4 developers incorporate coding semantics of other languages into their Ada programs that are inappropriate to Ada.

Corresponding characteristics include abbreviated identifiers, unstructured control features, and lack of effective problem modeling and abstraction due to the absence of appropriate data structures. Overall program design lacks modularity, utilizes excessive amounts of global data structures, and fails to control visibility of objects with the information hiding techniques of package structuring. The characteristics of Level 4 software defeat the purpose of requiring the Ada programming language for program development. A possible exception here is to allow developers to get started with Ada for hands-on training. However, in this case, developers must learn proper Ada structure very quickly.

Level 3 - "Programmed In Ada"

Level 3 represents the lowest acceptable criteria for justifying the existence of software in Ada form. The developers are properly trained in the basic principles of the language and know how to effectively utilize its features for developing readable and maintainable software. The software requirements are known and understood with a significant amount of pre-implementation thought going into the design of the program structure.

Level 3 programs have meaningful identifier names, use only structured programming constructs, and accurately model real-world objects with appropriate data structures. Program structure is highly modularized with inter-module coupling minimized and internal module structure strongly cohesive. Packages are properly used to support principles of information hiding, object encapsulation, and abstract data types. Visibility of objects is strongly controlled, data is strongly typed, and use of global objects is strictly limited.

Level 2 - "Designed Into Ada"

This level of quality concentrates on issues above the programming language level. A software design approach is adopted to properly define the structure of the modules of the software system independently of the implementation details of the target programming language. One or more design methodologies may be used to create consistency and reliability in the program structure. Since Ada directly supports the principles of good software design, an Ada-based Program Design Language (PDL) is very appropriate. However, the main idea is that the software system is specified and verified to a large degree prior to the implementation phase, at which point problems and errors are much more costly to correct.

The main characteristic of Level 2 software is that the overall software system design displays a very understandable structure that allows reliable modifications and enhancements. Software design documents are produced as deliverable products prior to program source code development. The design methodologies may be supported by automated tools that help verify interface consistency and requirements completeness. The actual source code programs resulting from the software design display all of the quality attributes associated with Level 3 software. Consequently, Level 2 software is more reliable, understandable, and more easily adapted to new applications.

Level 1 - "Engineered With Ada"

This classification corresponds to the highest degree of quality possible in Ada-based software. The software is created with a comprehensive software life-cycle approach by developers who are well trained and knowledgeable in the goals and principles of software engineering. The main emphasis in the process is in the distinction between the problem domain and the solution domain of the computer-based solution. The requirements analysis phase of development is utilized to fully understand the problem space and to determine exactly what the software is to do in the first place. A variety of methodologies and technologies may be used to ensure that valid requirements are specified up front and that the associated costs and risks are reduced. The analysis phase may include utilization of techniques such as rapid prototyping and higher-level applications generators for defining and refining user interface and system requirements, and for generating feedback from the user community. The remaining phases of design, implementation, testing, and debugging are all in the solution space of the development process and are concerned with how to meet the requirements specification.

Software that is engineered with Ada strongly supports the goals and principles of software engineering. Analysis is the main key to understanding which components of the software design actually need to be developed from scratch and which ones can be satisfied by existing reusable components. A very coherent and useable set of documentation is produced in the engineering process relating to the various phases of the life cycle, as well as documentation applicable to all phases of development. The concept of a project data or object base is realized and implemented for accurate control and accountability of personnel, products, and organizational information. Automated support tools are effectively utilized throughout all forms of development to increase productivity, support proper and disciplined development, and to reduce the manual effort required from software developers. And finally, an intense concern for maintainability is prevalent throughout all decision-making and phases of development.

4.0 USE OF HIGHER-LEVEL SPECIFICATION LANGUAGES

~~4.0- USE OF HIGHER-LEVEL SPECIFICATION LANGUAGES~~

It is difficult to assess the quality of Ada code that is automatically generated from a higher level of specification. The concept of an executable requirements or design language does support software engineering principles. However, the issue of quality rests in the question of what level of specification will the software be maintained at. If it is strictly at the higher level of requirements or design specification, then the actual source code generated will not be visible to the human programmer, and thus its structure will not be of great significance. However, if the resulting Ada code will be subject in any way to human analysis and subsequent modification, then the level of quality will be directly related to the same factors associated with well-engineered and manually-written Ada programs.

Therefore, in this latter case, the attractive process of generating Ada source code from a higher level of specification must be designed such that the corresponding characteristics and attributes associated with the resulting code coincide with those indicative of well-written Ada software developed directly by a human programmer. The degree of quality associated with the higher-level specification will consequently be based on the degree to which the automatically generated code displays the good human engineering principles needed for understandable and maintainable software.

ORIGINAL PAGE IS
OF POOR QUALITY

5.0 CONCLUSIONS

The usefulness of the preceding classification scheme for Ada-based software is highly dependent on a more precise and tangible definition of criteria for each class. Although this level of detail was not given, the taxonomy proposes a starting point from which to base further analysis. The main idea of the paper is to create an awareness of the potential problems to expect when transitioning to a new programming language such as Ada. The Ada language alone cannot solve the problems currently prevalent in large organizations such as NASA in which software costs are a significant portion of the budget. Ada, and its corresponding support environment, merely provide the best available set of tools which support and encourage the adherence to the proven and solid principles of software engineering.

The mandate for the Space Station Program to move into the "Ada culture" will be totally ineffective if engineering principles and corresponding methodologies are not properly utilized. Obviously, education and training will be essential for developing a smooth transition into the software engineering discipline. The spectrum of potential Ada software quality classes presented here can help create and maintain the awareness and importance of viewing software engineering as a true engineering discipline. This recognition will be essential for the success of the up-coming proliferation of Ada-based software projects in the Space Station Program.